Reconfiguration Planning for Pivoting Cube Modular Robots

Cynthia Sung, James Bern, John Romanishin, and Daniela Rus

***Abstract*— In this paper, we present algorithms for self-reconfiguration of modular robots that move by pivoting. The modules are cubes that can pivot about their edges along the $\hat{x}$, $\hat{y}$, or $\hat{z}$ axes to move on a 3-dimensional substrate. This is a different model from prior work, which usually considers modules that slide along their faces. We analyze the pivoting cube model and give sufficient conditions for reconfiguration to be feasible. In particular, we show that if an initial configuration does not contain any of three subconfigurations, which we call rules, then it can reconfigure into a line. We provide provably correct algorithms for reconfiguration for both 2-D and 3-D systems, and we verify our algorithms via simulation on randomly generated 2-D and 3-D configurations.**

I. Introduction

Modular robots consist of multiple connected modules, each with limited capabilities, that can be reconfigured to produce complex functionality as required by a task. Among the self-reconfigurable modular systems that have been developed [1]–[7], pivoting has emerged as a simple but powerful motion predicate [7]. In this paper, we describe a model for pivoting cubes in 3-D. We consider reconfiguration in both 2-D and 3-D settings and demonstrate that barring certain subconfigurations allows us to guarantee reconfiguration in $O(n^2)$ moves. We provide provably correct algorithms for performing such reconfiguration. These are not optimal but are the first correct algorithms for the 3-D pivoting cube model. We perform simulations on random 2-D and 3-D configurations and show that in many cases, reconfiguration does not require the upper bound of n^2 moves.

Pivoting modules, although prevalent in hardware [5], [7], are not well-studied. Pivoting modules sweep out a volume that extends past their initial and final positions, and any reconfiguration algorithm must take this motion constraint into account. An $O(n^2)$ algorithm for 2-D pivoting modules was given in [8] but the model relaxed connectivity constraints compared to what many modular robots require and allowed movements that are often not physically realizable. Nguyen et al. [9] considered 2-D pivoting hexagons while requiring strong connectivity (connectivity through faces) and provided sufficient conditions for reconfiguration in $O(n^{5/2})$ moves. In [10], pivoting squares were reconfigured using a stochastic approach, and work in [5] analyzed the same system using meta-modules. A model for pivoting cubes in 3-D was introduced in [7], but the planning problem was not addressed. To our knowledge, there have been no studies of reconfiguration for pivoting cubes in 3-D.

C. Sung, J. Romanishin, and D. Rus are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA emails: {crsung, johnrom, rus}@csail.mit.edu. J. Bern is with the Department of Mechanical Engineering, California Institute of Technology, Pasadena, CA, USA email: jbern@caltech.edu

In contrast, in sliding models, modules move by translating along the faces of adjacent modules. When strong connectivity is enforced, reconfiguration for 2-D shapes can be completed in $O(n^2)$ moves [11]–[13], or in $O(n)$ rounds for synchronous distributed settings [14]. 3-D versions of the problem have also been studied under the restriction that the initial and final configurations have no holes [15]. In [16], the 3-D sliding cube model enabled locomotion over arbitrary obstacles. Unfortunately, because of the additional clearance required for pivoting modules, none of these algorithms can be used on the pivoting cube model.

The crystalline model [1], [17] enables shape metamorphosis via expansion and contraction of modules. 2-D versions of these modules reconfigure in $O(n \log n)$ moves and $O(\log n)$ time steps when grouped into 2×2 meta-modules [18]. Similarly, 3-D reconfiguration can be performed in $O(n)$ moves with $2 \times 2 \times 2$ meta-modules [19]. Again, these algorithms do not work on pivoting modules because of the clearance requirements associated with pivoting.

The paper is organized as follows. Section II provides a description of the pivoting cube model, and section III gives the problem definition. Sections IV and V introduce algorithms for reconfiguration in 2-D and 3-D, as well as proofs of correctness. Section VI contains results of simulations for random 2-D and 3-D configurations using the proposed algorithms. Finally, we conclude in section VII.

II. Pivoting Cube Model

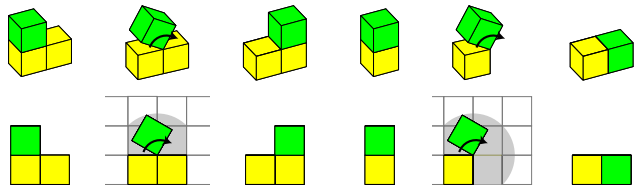
The pivoting cube model differs from other theoretical models in its motion constraints. Cubes locomote by pivoting (rotating) about the edges they share with other modules (ref. Fig. 1). Because a cube is longer along a diagonal than along a side, pivoting requires that cells adjacent to the initial and final positions also be empty. The result is that modules even with no neighbor along a face are sometimes unable to move. This is a significant difference from the sliding cube model.

In our model, a module pivots by the maximum angle possible until it contacts another module. If a module does not share an edge with another module parallel to the pivot axis, it will not pivot.

The model in this paper makes the following assumptions:

- A pivoting module rotates about an edge that it shares with another module (the pivot edge).
- A pivoting module sweeps out a volume that must not intersect other modules.
- This swept volume lies within one layer (along the pivot plane), which is perpendicular to the pivot edge.
- Except during pivoting, modules sit on a cubic lattice.

In addition, we enforce strong connectivity, that is



(a) Pivot by $\frac{\pi}{2}$ (b) Pivot by π

Fig. 1. Pivot moves that a module can perform. The green module pivots about an edge until it comes into contact with another module. The shaded gray area indicates the sweep volume of the pivot move. All grid cells containing gray must be empty in order for the module to pivot.

- Modules that are connected must share a face.

Since a pivoting module need share only the pivot edge with another module, the two are not necessarily connected.

III. Definitions and Problem Definition

A *configuration* $\mathcal{C}$ consists of n unit cube modules i on a cubic lattice, in which modules intersect either at a face, at an edge, or not at all. We assign a global coordinate system so that modules are positioned at integer coordinates (x_i, y_i, z_i). Each position in the lattice is called a *cell*, and it can be either empty (no module at that location) or occupied.

Two modules i and j are *connected* if they share a face, i.e., if they share two coordinates and the third is different by only one. In this case, module j is called module i's neighbor. The *connectivity graph* of a configuration $\mathcal{C}$ is a lattice graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V} = \{i : i \text{ is a module in } \mathcal{C}\}$ and edges $\mathcal{E} = \{(i, j) : i \text{ and } j \text{ are neighbors}\}$. A configuration is called connected if its connectivity graph is connected. The complement of $\mathcal{G}$ is the connectivity graph of the free space. It contains one or more components, one of which is unbounded. We define the boundary $\mathcal{O}$ of configuration $\mathcal{C}$ to be the set of modules $i \in \mathcal{C}$ that neighbor cells in the unbounded component. The bounded components are holes.

Certain modules on the boundary of a configuration are extreme modules, which are modules with minimum or maximum x, y, or z coordinate values. We identify them by the directions in which they are extreme. For example, the $(+\hat{z}, +\hat{y}, -\hat{x})$ extreme module is found by considering the modules with maximum z coordinate and out of those with the maximum y coordinate, taking the one with minimum x.

A. Reconfiguration

Modules can pivot about the edges of other modules to change the configuration. If a module in a connected configuration is able to pivot without disconnecting the remaining configuration even temporarily, then we say the module is *mobile*. The *mobile set* $\mathcal{M}$ is the set of all modules $i \in \mathcal{C}$ that are mobile. We assume that modules pivot in lockstep, and we discretize time according to the number of steps $t = 0, 1, \cdots$. Reconfiguration, then, is the problem of finding a sequence of configurations $(\mathcal{C}_0, \mathcal{C}_1, \cdots, \mathcal{C}_T)$ from an initial configuration to a goal configuration such that the transitions between the configurations at each time step consist of only valid, non-colliding pivots. That is:

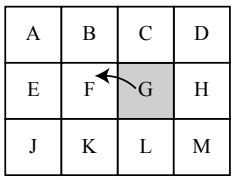


Fig. 2. Pivot plane of a module pivoting from cell G.

Problem 3.1 (Reconfiguration): Given two connected configurations $\mathcal{C}_0$ and $\mathcal{C}_{goal}$ both of size n, find a sequence $(\mathcal{C}_0, \mathcal{C}_1, \cdots, \mathcal{C}_T)$, subject to:

- *Termination.* $\mathcal{C}_T = \mathcal{C}_{goal}$
- *Validity.* At each $t \in \{1, \cdots, T\}$, $\mathcal{C}_t$ is the result of a set of modules in $\mathcal{C}_{t-1}$ pivoting.
- *Collision.* At each $t \in \{1, \cdots, T\}$, the modules pivoting between $\mathcal{C}_{t-1}$ and $\mathcal{C}_t$ do not sweep overlapping volumes.
- *Connectivity.* The configuration remains connected through the entire transition from $\mathcal{C}_0$ to $\mathcal{C}_T$.

The problem is feasible if there exists such as sequence. The environment is assumed to be obstacle-free.

B. Reversibility

Reversibility is the property that the effect of a pivot can immediately be undone by another pivot; that is, that the sequence $(\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_0)$ is a valid sequence. Figure 1 shows the two possible pivot types in our model. Pivots by π that maintain connectivity are reversible since the pivoting module is connected to the same neighbor after pivoting as before, and the volume swept out during the move is the same forwards and backwards. Of greater interest is pivots by $\frac{\pi}{2}$.

Property 3.2: Pivoting by $\frac{\pi}{2}$ is reversible if the pivoting module has a neighbor in the pivot plane prior to the move.

Proof: Consider a module that pivots by $\frac{\pi}{2}$, and label the cells along the pivot plane as shown in Fig. 2. The module begins in cell G. The neighboring cells must satisfy:

- *Collision-free.* Cells B, C, and F are all empty.
- $\frac{\pi}{2}$ *stop.* One of cell E or K is occupied.
- *Pivot edge.* One of cell K or L is occupied.
- *Neighbor.* One of cell H or L is occupied.

When the module pivots, it moves to cell F. The conditions for reversibility of this pivot are:

- *Collision-free.* Cells B, C, and G are all empty.
- $\frac{\pi}{2}$ *stop.* One of cell H or L is occupied
- *Pivot edge.* One of cell K or L is occupied.

The collision and pivot edge conditions are the same in both forward and backward moves. If the pivoting module had a neighbor in pivot plane, then that neighbor satisfies the $\frac{\pi}{2}$ stop condition and the pivot is reversible. ■

In connected configurations, irreversible pivots occur when the pivoting module, which must have had a neighbor to be connected, had neighbors outside the pivot plane (Fig. 3).

With reversible pivots, we can simplify the original problem to the following:

Problem 3.3 (Reconfiguration into a Line): Given a connected configuration $\mathcal{C}_0$ of size n, find a sequence $(\mathcal{C}_0, \mathcal{C}_1, \cdots, \mathcal{C}_T)$, subject to:

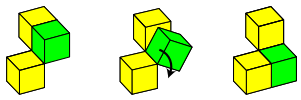
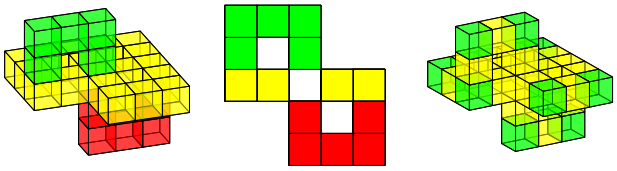
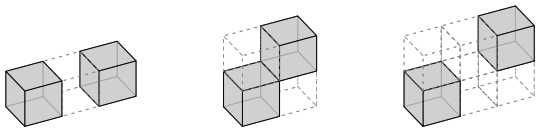


Fig. 3. Irreversible $\frac{\pi}{2}$ pivot. The two yellow modules are connected via the rest of the configuration (not shown). Since modules pivot until they contact another module, subsequently pivoting back would result in a pivot by π.



(a) Configuration (b) Center slice (c) Modules able to pivot

Fig. 4. Example of a configuration that cannot reconfigure. (b) View of center slice. (c) Modules that are able to pivot are highlighted in green. Pivoting any of these modules will disconnect the configuration.



(a) Rule 1 (b) Rule 2 (c) Rule 3

Fig. 5. Inadmissible subconfigurations. Occupied cells are filled in solid gray. Empty cells are outlined in dotted lines.

- *Termination.* $\mathcal{C}_T$ is a line.
- *Validity.* At each $t \in \{1, \cdots, T\}$, $\mathcal{C}_t$ is the result of a set of modules in $\mathcal{C}_{t-1}$ performing *reversible* pivots.
- *Collision.* At each $t \in \{1, \cdots, T\}$, the modules pivoting between $\mathcal{C}_{t-1}$ and $\mathcal{C}_t$ do not sweep overlapping volumes.
- *Connectivity.* The configuration remains connected through the entire transition from $\mathcal{C}_0$ to $\mathcal{C}_T$.

A solution to this problem implies a solution to Problem 3.1 since if two configurations $\mathcal{C}_0$ and $\mathcal{C}_{goal}$ can each reconfigure into lines, then $\mathcal{C}_0$ can reconfigure into $\mathcal{C}_{goal}$ by first reconfiguring into a line and then performing the sequence for $\mathcal{C}_{goal}$ in reverse.

C. Admissible Configurations

There exist configurations where no module is mobile (ref. Fig. 4). To enable reconfiguration, we introduce inadmissible subconfigurations, shown in Fig. 5, which we call rules. Configurations that contain no instance of a rule as a subconfiguration, up to rotation and mirroring, are said to obey that rule. It was shown in [9] that if a hexagonal lattice configuration obeys the hexagonal equivalent of rule 1, then there exists a mobile cube on the boundary. Since rule 1 is not sufficient to guarantee this property in a cubic lattice (ref. Fig. 6), we include two additional rules. Connected configurations that obey all three rules are called *admissible* and can reconfigure into lines.

IV. Reconfiguration in 2-D

Two-dimensional configurations are those with constant x, y, or z coordinate. For simplicity of notation, we assume that all modules have the same z coordinate and write the position of a module as (x_i, y_i). We also restrict modules to pivot only about $+\hat{z}$ or $-\hat{z}$, which we call pivoting counterclockwise and clockwise respectively. Under this restriction, all pivots are reversible as long as the configuration is connected. A description of the algorithm and proof of correctness follow.

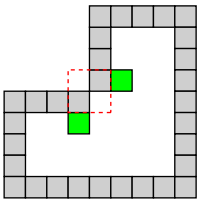


Fig. 6. 2-D configuration that obeys rule 1 but has no mobile module on the boundary. Mobile modules are green, boundary modules are gray. The red dotted line indicates an instance of rule 2.

Algorithm 1: RECONFIGURE2D($\mathcal{C}_0$)

Input: An admissible initial configuration $\mathcal{C}_0$

1 $T \leftarrow 0$; $q_i \leftarrow$ empty queue
2 $e \leftarrow (+\hat{y}, +\hat{x})$ extreme module of $\mathcal{C}_0$
3 $\mathcal{N} \leftarrow$ modules in $\mathcal{O} \cap \mathcal{M}$ in counterclockwise order starting from (x_e, y_e), excluding e
4 **while** *not a line* **do**
`// Next module to pivot (Alg. 2)`
5 $(i, q_i) \leftarrow$ NEXTMODULE$(q_i, \mathcal{C}_T, \mathcal{N})$
`// Move module to the tail`
6 **repeat**
7 $\mathcal{C}_{T+1} \leftarrow \mathcal{C}_T$ with module i pivoted clockwise
8 $T \leftarrow T + 1$
9 **until** $x_i = x_e$ *AND* $y_i > y_e$ `// i is extreme`
10 Update $\mathcal{N}$
11 **end**
12 **return** $(\mathcal{C}_t)_{t=0}^{T}$

A. Algorithm Description

Our algorithm is similar to that in [9]. It takes as input an admissible configuration $\mathcal{C}_0$ and reconfigures it into a line as shown in Algorithm 1. The algorithm works by finding the $(+\hat{y}, +\hat{x})$ extreme module and building a tail off of it in $+\hat{y}$ direction. Individual modules are selected greedily (line 5) using the procedure in Algorithm 2 and pivot clockwise until they reach the end of the tail (lines 6–9). In particular, to find the next module to pivot to the end of the tail, we start at the extreme module and search the boundary $\mathcal{O}$ of the configuration in a counterclockwise direction for the first mobile module i such that either (type 1) module i can be removed from the configuration without rendering it inadmissible, or (type 2) there exists a trio of modules (i, j, k) that form part of an instance of P_3, depicted in Fig. 7. In the case of a type 1 module, the module i pivots to the end of the tail, and the search process restarts. In case of type 2 modules, all three modules i, j, and k pivot to the end of the tail in that order before a new search begins.

Algorithm 2: NEXTMODULE$(q_i, \mathcal{C}, \mathcal{N})$

Input: A queue q_i of modules, an admissible configuration $\mathcal{C}$, and a list $\mathcal{N}$ of mobile boundary modules in counterclockwise order

1 **if** q_i *is not empty* **then**
    `// Choose next module in the queue`
2 $i \leftarrow q_i$.pop()
3 **else**
4 $i \leftarrow$ first module in $\mathcal{N}$ such that $\mathcal{C} \setminus i$ admissible OR i in an instance of P_3
5 **if** $\mathcal{C} - i$ *not admissible* **then**
        `// Type 2`
6 Find j, k such that (i, j, k) in an instance of P_3
7 q_i.push(j)
8 q_i.push(k)
9 **end**
10 **end**
11 **return** (i, q_i)

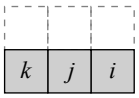


Fig. 7. Subconfiguration P_3. Occupied cells are filled in solid gray. Empty cells are outlined in dotted lines.

B. Analysis

To show correctness of this algorithm, we make use of the following lemmas.

Lemma 4.1: A mobile module i pivoting on the boundary of an admissible configuration $\mathcal{C}$ is able to traverse around the entire boundary.

Proof: Module i is able to pivot at least once by definition. Furthermore, a module that has just pivoted is able to continue pivoting in the same direction. Consider Fig. 2. Without loss of generality, assume a module in the gray cell G just pivoted $\frac{\pi}{2}$ clockwise from cell F. (Note that a module pivoting by π must pass through cell F.) In order for the module to have pivoted to cell G, cells B and C must be empty. Since the module pivoted from cell F, cell F must now be empty. Since cells B, C, and F are empty, regardless the state of other cells, the module is able to pivot clockwise again. Thus a module is only unable to continue pivoting if such a pivot would disconnect the configuration.

There are two ways for a pivot to disconnect a configuration: 1) the pivot disconnects other modules from each other, or 2) the pivot disconnects the pivoting module. The second case is prohibited by rule 1. As for the first, when the module pivoted initially, the configuration must have been connected by definition of mobile. The module cannot be necessary for maintaining connectivity between other modules, no matter how many times it has pivoted.

So a mobile module is able to pivot around part of the boundary of $\mathcal{C}$ and return to its original position. Furthermore, since $\mathcal{C}$ obeys rule 3, the module cannot 'skip over' parts of the boundary. That is, neighbors of the module's initial and final positions must be neighbors of each other. The module is able to traverse around the entire boundary of the configuration. ■

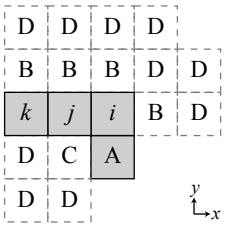


Fig. 8. Figure for Lemma 4.2. Neighboring cells of module i. Occupied cells are filled in solid gray. Empty cells are outlined in dotted lines.

Since the tail is part of the boundary, Lemma 4.1 indicates that any mobile cube is able to pivot to the end of the tail.

Lemma 4.2: Given an admissible configuration $\mathcal{C}$ that is not a line, there exists a mobile module i that satisfies the search criteria in Algorithm 2, line 4.

Proof: Either there exists a mobile module $i \in \mathcal{O}$ of type 1 or not. If it exists, then this module satisfies the criteria. If it does not, we show that there must exist an instance of P_3 on the boundary.

Let $\mathcal{B}$ be the block tree of the connectivity graph $\mathcal{G}$ of $\mathcal{C}$ and take only the subgraph $\mathcal{B}_\mathcal{O} \subseteq \mathcal{B}$ where every block contains a module in the boundary $\mathcal{O}$. Choose an arbitrary leaf $b \in \mathcal{B}_\mathcal{O}$. Unless $\mathcal{B}_\mathcal{O}$ is a single vertex, b contains a module ℓ that is also contained in a different block of $\mathcal{B}_\mathcal{O}$ (ℓ is a cut vertex in the connectivity graph of $\mathcal{C}$). Since b is a leaf, there can only be one, so all modules on the boundary of b other than ℓ must also be in $\mathcal{O}$. Furthermore, the boundary of b must form a simple cycle; otherwise there exists a boundary module with only one neighbor and removing that module from the configuration would leave the configuration admissible, which contradicts our original assumption.

Of the extreme modules of b, choose a module i that is not the extreme module of $\mathcal{C}$ and that has a different x coordinate from ℓ. Say this module is the $(+\hat{y}, +\hat{x})$ extreme. The neighboring cells are depicted in Fig. 8. Since the boundary of b is a cycle and i is on the boundary, i must have two neighbors (cells labeled j and A). Furthermore, since i is extreme and has a different x coordinate from ℓ, there can be no modules in the cells labeled B. There can be no module at C, or else $\mathcal{C} - i$ would be admissible, which contradicts our original assumption. Therefore, there is a module at k. The cells labeled D must be empty since $\mathcal{C}$ is admissible. Hence the modules at (i, j, k) form an instance of P_3. ■

Lemma 4.3: Pivoting a type 1 mobile module to the end of the tail of an admissible configuration $\mathcal{C}$ produces an admissible configuration.

Proof: Pivoting a module to the end of the tail has the same effect as removing it from the original configuration $\mathcal{C}$ and adding a new module to the end of the tail. Adding new modules to the tail does not affect the admissibility of $\mathcal{C}$ since the tail is grown in the $+\hat{y}$ direction off the $(+\hat{y}, +\hat{x})$ extreme module and there can exist no other modules with

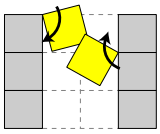


Fig. 9. Two modules arbitrarily far apart on the boundary of an admissible configuration may still collide.

a greater y coordinate. Removing a type 1 module from $\mathcal{C}$ does not render it inadmissible by definition of type 1. ■

Now consider type 2 mobile modules. Fig. 8 shows that modules i, j, and k are all on the boundary, module j is mobile once i pivots, and module k is mobile once j pivots. All three modules are able to pivot to the end of the tail.

Lemma 4.4: Pivoting a sequence of type 2 mobile modules to the end of the tail of an admissible configuration $\mathcal{C}$ produces an admissible configuration.

Sketch of Proof: As mentioned for Lemma 4.3, we only have to consider the effect of removing i, j, and k from the configuration. This does not render $\mathcal{C}$ inadmissible, as can be seen by checking Fig. 8 against the rules. ■

Combining these lemmas gives us our main result.

Theorem 4.5: Algorithm 1 reconfigures any admissible 2-D configuration of size n into a straight line configuration using $O(n^2)$ pivot moves.

Proof: Correctness: For any admissible configuration $\mathcal{C}_0$ that is not a line, Lemma 4.2 states that Algorithm 2 will return a module i. By Lemma 4.1, the module can pivot to the end of the tail, and by Lemmas 4.3 and 4.4, these pivots do not affect the admissibility of the configuration. Therefore, mobile modules will continue to be identified and pivot to the tail until the configuration is a straight line.

Pivot Moves: Up to n modules must pivot to the end of the tail, and each pivots up to n times. The total number of pivot moves is at most n^2. ■

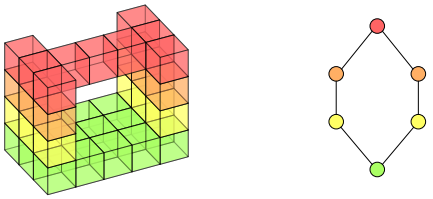
This bound is tight since $\Theta(n^2)$ moves are required to reconfigure a horizontal line into a vertical line.

Sometimes, a configuration is rooted at a module that cannot pivot and is thus immobile. In this case, all lemmas hold except Lemma 4.2. Because of the root, certain modules between the root and the extreme module will also be rendered immobile. However, the block tree analysis holds, and the final configuration will not contain any cycles.

Corollary 4.6: Algorithm 1 reconfigures any admissible 2-D configuration with an immobile root into an admissible chain configuration with one end at the root.

C. Parallel Moves

In the proposed algorithm, one module pivots at a time. Allowing multiple modules to pivot simultaneously will speed up the process but also introduces the possibility of collisions between two pivoting modules. Even when a configuration is admissible, two modules arbitrarily far apart in distance along the boundary may collide if they pivot simultaneously (ref. Fig. 9). Mobile modules would have to keep track of the locations of all other moving modules in order to completely avoid collision. To avoid these situations, we can



(a) Configuration (b) Slice Graph

Fig. 10. Configuration and its corresponding slice graph. Each layer is colored in different color. The red and green slices are locally extreme.

restrict configurations such that any two pivot edges on the boundary are a distance more than $2\sqrt{2}$ units apart, which is the length of two diagonals. We call this the *parallelization criterion*. When this criterion is satisfied, it can be shown that modules pivoting more than 4 units apart on the boundary of a configuration are guaranteed not to collide. Using the same procedure as in Algorithm 1 but with each module waiting only until the previous one has advanced 8 units along the boundary before pivoting (pivoting modules may free modules up to 4 units ahead) yields the following result.

Corollary 4.7: An admissible configuration $\mathcal{C}$ of size n that satisfies the parallelization criterion can be reconfigured into a straight line in $O(n)$ time steps.

D. Reduced Space Requirements

Although we use the $(+\hat{y}, +\hat{x})$ extreme module to build the tail, any extreme module can be chosen. Practically, this allows some time and space savings for general reconfiguration. Rather than building a line of length n and then running the steps backwards to produce a goal configuration, modules that reach the tail of the initial configuration can continue to build the goal configuration starting at its $(-\hat{y}, +\hat{x})$ extreme. This decreases the width of the space required to be the sum of the widths of the initial and final configurations.

V. Reconfiguration in 3-D

Our algorithm for 3-D configurations is similar to that for 2-D configurations. In order to determine which modules to move to the tail, we divide the configuration into 2-D slices.

A *layer* of a configuration $\mathcal{C}$ is a cross-section $\mathcal{L} \subseteq \mathcal{C}$ with constant x, y, or z coordinate. For example, the 2-D $(z = 2)$-layer of $\mathcal{C}$ is all modules with z coordinate equal to 2, $\mathcal{L} = \{i \in \mathcal{C} : z_i = 2\}$. The 1-D $(x = 1, y = 0)$-layer is a line $\mathcal{L} = \{i \in \mathcal{C} : x_i = 1 \text{ AND } y_i = 0\}$. A *slice* is a maximal connected component of a layer.

Every 3-D configuration can be represented by a *slice graph* $\mathcal{G}_\mathcal{S} = (\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S})$, where $\mathcal{V}_\mathcal{S}$ contains all the slices of $\mathcal{C}$ cut along one axis and $\mathcal{E}_\mathcal{S} = \{(\mathcal{S}_1, \mathcal{S}_2) : \mathcal{S}_1 \text{ and } \mathcal{S}_2 \text{ contain neighboring modules}\}$. For this algorithm, we use slices cut along the $\hat{z}$ axis (Fig. 10). A slice is called *locally extreme* if either all neighboring slices have z coordinate less than its own or all neighboring slices have z coordinate greater than its own.

A. Algorithm Description

Similarly to the 2-D algorithm, modules are chosen sequentially and moved to a tail, which in this case grows

Algorithm 3: RECONFIGURE3D($\mathcal{C}_0$)

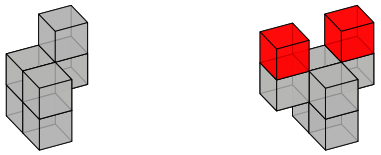
Input: An admissible initial configuration $\mathcal{C}_0$ whose holes are convex

1 $T \leftarrow 0$; $q_i \leftarrow$ empty queue
2 $e \leftarrow (+\hat{z}, +\hat{y}, +\hat{x})$ extreme module of $\mathcal{C}_0$
3 $\mathcal{G}_\mathcal{S} = (\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S}) \leftarrow$ slice graph of $\mathcal{C}_0$
4 **while** *not a line* **do**
5 Pick a slice $\mathcal{S}_i \in \mathcal{V}_\mathcal{S}$ to deconstruct
`// Deconstruct slice` $\mathcal{S}_i$
6 $e_\mathcal{S} \leftarrow (+\hat{y}, +\hat{x})$ extreme module of $\mathcal{S}_i$
7 $r_\mathcal{S} \leftarrow$ root module of $\mathcal{S}_i$
8 $\mathcal{N}_\mathcal{S} \leftarrow$ mobile modules on the boundary of $\mathcal{S}_i$ in counterclockwise order starting from $(x_{e_\mathcal{S}}, y_{e_\mathcal{S}}, z_{e_\mathcal{S}})$, excluding $e_\mathcal{S}$ and $r_\mathcal{S}$
9 **while** $\mathcal{S}_i$ *not empty* **do**
10 **if** *isempty*($\mathcal{N}_\mathcal{S}$) **then**
`// ` $\mathcal{S}_i$ `is a chain; choose the extreme module`
11 $i \leftarrow e_\mathcal{S}$
12 $e_\mathcal{S} \leftarrow$ extreme module of $\mathcal{S}_i - e_\mathcal{S}$
13 **else**
`// Next module (Alg. 2)`
14 $(i, q_i) \leftarrow$ NEXTMODULE($q_i, \mathcal{S}_i, \mathcal{N}_\mathcal{S}$)
15 **end**
`// Move module to the tail`
16 $P \leftarrow$ reversible path from (x_i, y_i, z_i) to tail
17 **repeat**
18 $\mathcal{C}_{T+1} \leftarrow \mathcal{C}_T$ with i pivoted one step along P
19 $T \leftarrow T + 1$
20 **until** $x_i = x_e$ *AND* $y_i = y_e$ *AND* $z_i > z_e$
21 Update $\mathcal{N}_\mathcal{S}$
22 **end**
23 $\mathcal{G}_\mathcal{S} \leftarrow \mathcal{G}_\mathcal{S} - \mathcal{S}_i$
24 **end**
25 **return** $(\mathcal{C}_t)_{t=0}^{T}$

in the $+\hat{z}$ direction off the $(+\hat{z}, +\hat{y}, +\hat{x})$ extreme module e. The full procedure is provided in Algorithm 3. The algorithm works by dividing the 3-D configuration into its 2-D slices and breaking down these slices sequentially.

a) Choosing a slice (line 5): The slice graph $\mathcal{G}_\mathcal{S}$ of the input configuration is constructed, and a slice $\mathcal{S}_i \in \mathcal{G}_\mathcal{S}$ is chosen to move to the end of the tail. Slices corresponding to tail modules are ignored. If there is more than one slice, $\mathcal{S}_i$ is chosen such that 1) $\mathcal{G}_\mathcal{S} - \mathcal{S}_i$ is connected, 2) the extreme module e is not in $\mathcal{S}_i$, and 3) $\mathcal{S}_i$ is locally extreme. Once the modules in the slice have pivoted to the end of the tail (we say the slice is *deconstructed*), a new slice is chosen. We call the movement of one slice to the end of the tail one *stage*. The last slice to move to the end of the tail is the slice containing the extreme module e.

b) Deconstructing a slice (lines 6–22): Since the configuration must remain connected throughout reconfiguration, a root module $r_\mathcal{S}$, which is a module with a neighbor not in $\mathcal{S}_i$, is marked immobile (line 7). All modules in the slice are then moved to the end of the tail in an order similar to the 2-D algorithm, leaving only a chain configuration behind. At this point, the chain is also moved to the tail, starting at the extreme module $e_\mathcal{S}$ of the slice and ending at the root (lines 10–12). Path planning for individual modules uses a breadth-first search starting at the end of the tail. For each cell (x, y, z), the reachable cells are found by simulating a module at (x, y, z) pivoting about $\pm\hat{x}$, $\pm\hat{y}$, and $\pm\hat{z}$. Only cells reachable by reversible pivots are kept. It can be shown that the module can reach at least 4 distinct positions.



(a) Convex (b) Nonconvex

Fig. 11. Convex vs nonconvex configuration. (b) is nonconvex because the slice containing the red modules is not connected.

B. Analysis

Algorithm 3 reconfigures any admissible configuration with only convex holes into a line. A hole is convex if every 1-D layer of the hole is connected (ref. Fig. 11). In configurations with convex holes, every 2-D slice of the configuration contains a module that is on the boundary.

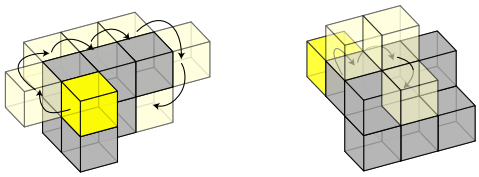
Lemma 5.1: Given an admissible configuration $\mathcal{C}$ with only convex holes, there exists a slice $\mathcal{S}_i$ that satisfies the selection criteria for line 5.

Sketch of Proof: When there is only one slice, that slice is chosen. Now consider if there are more slices. If there is a slice that does not contain the extreme module e and has one neighbor in $\mathcal{G}_\mathcal{S}$, this slice satisfies the criteria. Otherwise, consider the block graph $\mathcal{B}_\mathcal{S}$ of $\mathcal{G}_\mathcal{S}$. Similarly to the proof of Lemma 4.2, pick an arbitrary leaf block $b \in \mathcal{B}_\mathcal{S}$ (that does not contain e if there is more than one), and choose a locally extreme slice that is not also in a different block and does not contain e (there is at least one). This slice satisfies the selection criteria. ■

Lemma 5.2: The resulting configuration at the end of each stage is admissible and has only convex holes.

Proof: We prove this by induction. Assume the configuration is admissible with only convex holes at the end of one stage. Then this must also be true at the end of the next. Note that the initial configuration satisfies this condition.

Similarly to Lemma 4.3, the tail does not affect admissibility or the shape of any holes, so we only must check the removal of the slice. Since an entire slice is removed, rule 2 cannot be broken unless it was broken before the stage began, which would contradict the inductive hypothesis. Since the slice is locally extreme by our choosing, rules 1 and 3 also cannot be broken unless some rule was broken before. The slice was chosen to maintain connectivity of $\mathcal{G}_\mathcal{S}$, so the configuration remains connected and therefore admissible. Finally, removing an entire slice will not turn a convex hole



(a) (b)

Fig. 12. Paths to reach the end of the tail. The bright yellow module is the moving module. Light yellow cells indicate cells that the module passes through. (a) The module pivots about the boundary of its slice to the extreme position, then to the slice below. (b) The module pivots to the slice above, across the top of the slice, and into a cell in $\mathcal{G}_e$.

nonconvex. So the configuration must remain admissible and have only convex holes at the end of each stage. ■

Although the configuration is admissible between stages, it may not be admissible while a slice is in the process of deconstructing. During a stage, it is only necessary for a slice to be able to completely deconstruct (i.e., there exist paths to the end of the tail for every module in the slice).

Lemma 5.3: Let $\mathcal{C}$ be an admissible configuration with only convex holes at the start of a stage, and $\mathcal{S}_i$ be the slice chosen in line 5. Suppose m modules from $\mathcal{S}_i$ have pivoted to the end of the tail. If the configuration and $\mathcal{S}_i$ remain connected, then there exists a reversible path along which a mobile module i in $\mathcal{S}_i$ can pivot to the end of the tail.

Proof: Consider the configuration $\mathcal{C}$ with $\mathcal{S}_i$ removed. The result is admissible by Lemma 5.2. Suppose a module j is placed in an empty cell neighboring a module on the boundary. By Lemma 4.1 and connectivity of the slice graph of $\mathcal{C}$, module j is able to traverse around the entire boundary using only reversible moves. We can therefore construct a connected graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ where $\mathcal{V}_c$ contains all empty cells neighboring modules on the boundary, and $\mathcal{E}_c = \{(j,k) :$ a module at j can reach cell k via a reversible pivot$\}$.

Now take the connected slice $\mathcal{S}_i$, where m modules have already been removed. Delete from $\mathcal{G}_c$ any cells that are still occupied by modules in $\mathcal{S}_i$. Of the components of the resulting graph $\mathcal{G}_i$, exactly one contains the extreme module e. Call this component $\mathcal{G}_e$. If module i pivots to a cell in $\mathcal{G}_e$, then it can follow the paths in $\mathcal{G}_e$ to the end of the tail. We show that module i is able to reach a cell in that component.

Assume without loss of generality that $\mathcal{S}_i$ is locally extreme in the $+\hat{z}$ direction. The cells in $\mathcal{G}_i$ with neighbors in $\mathcal{S}_i$ must have a z coordinate of either z_i or $z_i - 1$. Furthermore, all cells with z coordinate equal to $z_i - 1$ belong to the same component; otherwise removing $\mathcal{S}_i$ would disconnect the configuration. Suppose $\mathcal{G}_e$ is this component. Then i is able to reach a cell in $\mathcal{G}_e$ by pivoting about $-\hat{z}$ to neighbor the extreme module of $\mathcal{S}_i$ (ref. Lemma 4.1), pivoting about $-\hat{x}$ to a cell with z coordinate $z_i - 1$, and then pivoting across the bottom of $\mathcal{S}_i$ to a cell in $\mathcal{G}_e$ (valid by rule 1). Fig. 12(a) shows an example of this path.

Suppose that $\mathcal{G}_e$ is a different component. Since $\mathcal{S}_i$ obeys rule 1, a module that neighbors a module in $\mathcal{S}_i$ is able to rotate about $\pm\hat{x}$ or $\pm\hat{y}$ to a position on top of the slice ($z = z_i + 1$). Since $\mathcal{C}$ was admissible, it is also able to cover the top of the slice by pivoting about $\pm\hat{x}$ and $\pm\hat{y}$. So, module i can pivot to the top of the slice, then across the top of the slice to a cell where it can pivot into a cell in $\mathcal{G}_e$ (Fig. 12(b)).

So as long as the configuration and $\mathcal{S}_i$ remain connected, a mobile module $i \in \mathcal{S}_i$ can pivot to the end of the tail. ■

Lemma 5.4: Let $\mathcal{C}$ be an admissible configuration with only convex holes and $\mathcal{S}_i$ be the slice chosen in line 5. The slice $\mathcal{S}_i$ can be deconstructed.

Sketch of Proof: According to Lemma 5.3, mobile modules in $\mathcal{S}_i$ can pivot to the end of the tail as long as $\mathcal{S}_i$ and $\mathcal{C}$ remain connected. Since modules are chosen according to Algorithm 2 (line 14), $\mathcal{S}_i$ remains connected. In addition, the root module $r_{\mathcal{S}}$ ensures that $\mathcal{C}$ remains connected. Therefore, mobile modules in $\mathcal{S}_i$ will pivot to the end of the tail as long as Algorithm 2 returns a module i. According to Corollary 4.6, this will occur until $\mathcal{S}_i$ is a chain starting at $r_{\mathcal{S}}$ and ending at $e_{\mathcal{S}}$. When $\mathcal{S}_i$ is a chain, $\mathcal{G}_i$ is one connected component, and $e_{\mathcal{S}}$ is able to pivot to the end of the tail in a process similar to that described in the proof of Lemma 5.3, leaving a slice rooted at $r_{\mathcal{S}}$ and ending at a new extreme module. Continuing in this manner leaves $\mathcal{S}_i$ and $\mathcal{C}$ connected, until $r_{\mathcal{S}}$ becomes the sole module in $\mathcal{S}_i$ and pivots to the end of the tail along a path in $\mathcal{G}_c$. ■

Combining these lemmas yields our main result:

Theorem 5.5: Given an admissible 3-D configuration $\mathcal{C}_0$ of size n whose holes, if any, are all convex, Algorithm 3 reconfigures $\mathcal{C}_0$ into a straight line configuration. Moreover, the algorithm terminates in $O(n^2)$ pivot moves.

Proof: Correctness: By Lemma 5.1 and 5.2, for any admissible configuration with only convex holes that is not a line, there exists a slice such that removal of the slice keeps the configuration admissible and all holes convex. By Lemma 5.4, that slice is able to be completely deconstructed. Therefore, slices will continue to be identified and modules moved to the tail until the configuration is a straight line.

Pivot moves: Up to n modules pivot up to n times, for a total of at most n^2 pivot moves. ■

All the pivot moves are reversible, so this algorithm also solves the general reconfiguration problem for admissible 3-D configurations with convex holes. We conjecture that a similar algorithm in which slices are partially deconstructed to open up holes will also work for nonconvex holes.

VI. Simulation Results

We have implemented the algorithms and simulated them on 1000 random admissible configurations of size varying from 5 modules to 110 modules. All runs successfully reconfigured the starting configuration into a line. Examples can be found in the accompanying video. Figures 13 and 14 show the number of time steps to reconfigure random 2-D and 3-D configurations respectively. The upper bound of n^2 is shown for reference. For nonparallel algorithms, the number of time steps equals the total number of moves. In both the 2-D and 3-D case, this number grew as n^2, although no configurations achieved the worst case estimate. We also implemented and simulated the 2-D algorithm with parallel

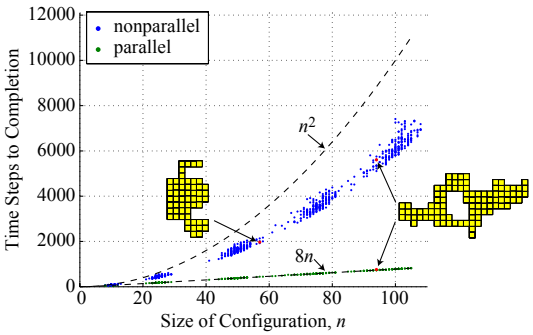


Fig. 13. Number of moves required for reconfiguration of random 2-D admissible configurations using the basic and parallel 2-D algorithms, with example configurations overlaid. While the total number of moves required is quadratic in n, the parallel algorithm can complete in $O(n)$ time steps.

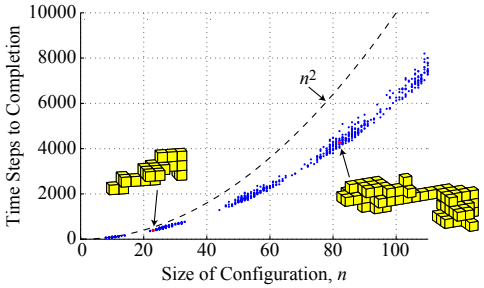


Fig. 14. Number of moves required for reconfiguration of random 3-D admissible configurations with convex holes. The upper bound of n^2 moves is indicated by a dotted line. Example random configurations are also shown.

moves, which completes in a linear number of time steps as shown in Fig. 13. In no simulation did modules collide.

VII. Discussion

In this paper, we consider reconfiguration for modular robots under the pivoting cube model. We present three rules that are sufficient to guarantee feasibility of reconfiguration for 2-D configurations and for 3-D configurations with convex holes, and we provide algorithms for reconfiguration into a line in $O(n^2)$ pivot moves. We also parallelize the 2-D algorithm to complete in $O(n)$ time steps. Our algorithms are provably correct and verified in simulations.

Our algorithms guarantee reconfiguration under certain conditions, but as mentioned in Section III-C, there exist configurations for which reconfiguration is infeasible. We are currently characterizing these types of configurations and investigating the class of configurations for which reconfiguration is feasible but may not be solved using greedy approaches. In addition, the algorithms proposed here are centralized and nonoptimal. Although we demonstrated that the 2-D algorithm could be parallelized, such parallelization requires imposing further constraints on the configuration, and individual modules must still use global knowledge to check whether the configuration is admissible to start. For systems with large numbers of modules, it is preferable that modules make decisions about pivoting using only local information. Future work includes distributed approaches to reconfiguration with the pivoting cube mode, as well as more optimal solutions to the general reconfiguration problem.

Acknowledgments

Support for this project has been provided in part by NSF Grant Nos. 1240383 and 1138967, by the DoD through the NDSEG Fellowship Program, by the Caltech SURF program, and by the Rose Hills Foundation. We would like to thank Kyle Gilpin, Thomas Bertossi, Nancy Lynch, Mira Radeva, Sharon Paradesi, Erik Demaine, Jingjin Yu, and Sebastian Claici for helpful feedback and discussions.

References

[1] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Autonomous Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[2] D. Brandt and D. J. Christensen, "A new meta-module for controlling large sheets of atron modules," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2007, pp. 2375–2380.

[3] K. Gilpin, A. Knaian, and D. Rus, "Robot pebbles: One centimeter modules for programmable matter through self-disassembly," in *IEEE Intl. Conf. on Robotics and Automation*, 2010, pp. 2485–2492.

[4] J. Davey, N. Kwok, and M. Yim, "Emulating self-reconfigurable robots - design of the smores system," in *IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*, 2012.

[5] P. J. White and M. Yim, "Reliable external actuation for full reachability in robotic modular self-reconfiguration," *Intl. Journal of Robotics Research*, vol. 29, no. 5, pp. 598–612, 2010.

[6] Y. Meng, Y. Zhang, A. Sampath, Y. Jin, and B. Sendhoff, "Cross-Ball: A new morphogenetic self-reconfigurable modular robot," in *IEEE Intl. Conf. on Robotics and Automation*, 2011, pp. 267–272.

[7] J. W. Romanishin, K. Gilpin, and D. Rus, "M-Blocks: Momentum-driven, magnetic modular robots," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2013.

[8] N. M. Benbernou, "Geometric algorithms for reconfigurable structures," Ph.D. dissertation, Massachusetts Institute of Technology, 2011.

[9] A. Nguyen, L. J. Guibas, and M. Yim, "Controlled module density help reconfiguration planning," in *4th Intl. Workshop on Algorithmic Foundations of Robotics*, 2000, pp. 23–36.

[10] N. Ayanian, P. J. White, A. Hálász, M. Yim, and V. Kumar, "Stochastic control for self-assembly of XBots," in *ASME Intl. Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2008, pp. 1169–1176.

[11] C.-J. Chiang and G. S. Chirikjian, "Modular robot motion planning using similarity metrics," *Autonomous Robots*, vol. 10, no. 1, pp. 91–106, 2001.

[12] A. Dumitrescu and J. Pach, "Pushing squares around," *Graphs and Combinatorics*, vol. 22, no. 1, pp. 37–50, 2006.

[13] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff, "Evaluating efficiency of self-reconfiguration in a class of modular robots," *Journal of Robotic Systems*, vol. 13, no. 5, pp. 317–338, 1996.

[14] J. E. Walter, J. L. Welch, and N. M. Amato, "Concurrent metamorphosis of hexagonal robot chains into simple connected configurations," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 6, pp. 945–956, 2002.

[15] D. Pickem, M. Egerstedt, and J. S. Shamma, "Complete heterogeneous self-reconfiguration: Deadlock avoidance using hole-free assemblies," in *4th IFAC Workshop on Distributed Estimation and Control in Networked Systems (NecSys)*, 2013.

[16] R. Fitch and Z. Butler, "Scalable locomotion for large self-reconfiguring robots," in *IEEE Intl. Conf. on Robotics and Automation*, 2007, pp. 2248–2253.

[17] D. Rus and M. Vona, "Self-reconfiguration planning with compressible unit modules," in *IEEE Intl. Conf. on Robotics and Automation*, vol. 4, 1999, pp. 2513–2520.

[18] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. Wuhrer, "Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves," in *Algorithms and Computation*, 2008, pp. 342–353.

[19] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, and S. Wuhrer, "Linear reconfiguration of cube-style modular robots," *Computational Geometry*, vol. 42, no. 6, pp. 652–663, 2009.